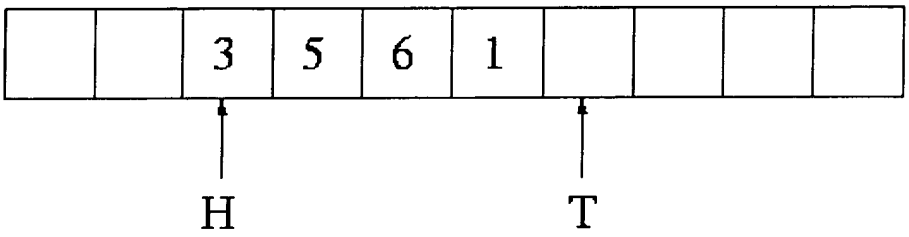
1.4. Circular queues

A circular queue is a particular implementation of a queue. It is very efficient. It is also quite useful in low level code, because insertion and deletion are totally independant, which means that you don't have to worry about an interrupt handler trying to do an insertion at the same time as your main code is doing a deletion.

How does it work?

A circular queue consists of an array that contains the items in the queue, two array indexes and an optional length. The indexes are called the *head* and *tail* pointers and are labelled H and T on the diagram.



The head pointer points to the first element in the queue, and the tail pointer points just beyond the last element in the queue. If the tail pointer is before the head pointer, the queue wraps around the end of the array.

Is the queue empty or full?

There is a problem with this: Both an empty queue and a full queue would be indicated by having the head and tail point to the same element. There are two ways around this: either maintain a variable with the number of items in the queue, or create the array with one more element that you will actually need so that the queue is never full.

Operations

Insertion and deletion are very simple. To insert, write the element to the tail index and increment the tail, wrapping if necessary. To delete, save the head element and increment the head, wrapping if necessary. Don't use a modulus operator for wrapping (`mod` in Pascal, `%` in C) as this is very slow. Either use an `if` statement or (even better) make the size of the array a power of two and simulate the `mod` with a binary `and` (`&` in C).

[Prev] [Next] [Up]